Lexical Analysis

Input: ***Program represented by a sequence of characters***

Tasks:	**Compiler modul:**
	Input reader
Recognize and classify tokens	Scanner (central phase, finite state machine)
Skip irrelevant characters	
Encode tokens:	
	Identifier modul
Store token information	Literal modules
Conversion	String storage

Output: ***Program represented by a sequence of encoded tokens***

© 2001 bei Prof. Dr. Uwe Kastens

Representation of tokens

Uniform encoding of tokens by triples:

Syntax code	**attribute**	**source position**
terminal code of the concrete syntax	value or reference into data module	to locate error messages of later compiler phases

Examples:

```
double sum = 5.6e-5;
while (count < maxVect)
{ sum = sum + vect[count];
```

DoubleToken		12, 1
Ident	138	12, 8
Assign		12, 12
FloatNumber	16	12, 14
Semicolon		12, 20
WhileToken		13, 1
OpenParen		13, 7
Ident	139	13, 8
LessOpr		13, 14
Ident	137	13, 16
CloseParen		13, 23
OpenBracket		14, 1
Ident	138	14, 3

© 2001 bei Prof. Dr. Uwe Kastens

Specification of token notations

Example: identifiers

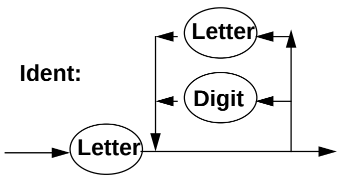
Ident = Letter (Letter | Digit)*

regular grammar ⟷ **regular expression**

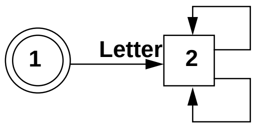
syntax diagram

Ident ::= Letter X
X ::= Letter X
X ::= Digit X
X ::=

Ident:



acceptor: **finite state machine**

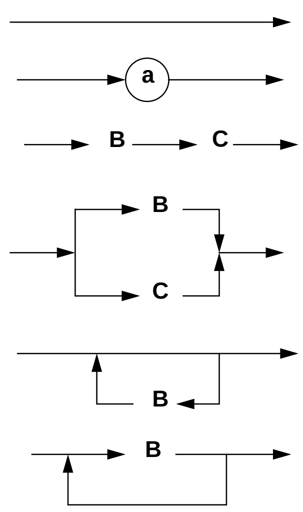


© 2001 bei Prof. Dr. Uwe Kastens

Regular expressions mapped to syntax diagrams

Transformation rules:

regular expression A — **syntax diagram for A**



regular expression A	
empty	**empty**
a	**single character**
B C	**sequence**
B | C	**alternative**
B*	**repetition, may be empty**
B$^+$	**repetition, non-empty**

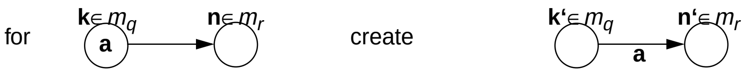
© 2001 bei Prof. Dr. Uwe Kastens

Construction of deterministic finite state machines

Syntax diagram	**deterministic finite state machine**
nodes, arcs	transitions, states
set of nodes m_q	state q
sets of nodes m_q and m_r connected with the same character a	transitions q ---> r with character a

Construction:

1. **enumerate nodes**; exit of the diagram gets the number *0*
2. **initial set of nodes** m_1 contains all nodes that are reachable from the begin of the diagram — **initial state** *1*
3. **construct new sets of nodes (states)** and **transitions:** For a character a and a set m_q containing node k create set m_r with all nodes n, according to the following schema:

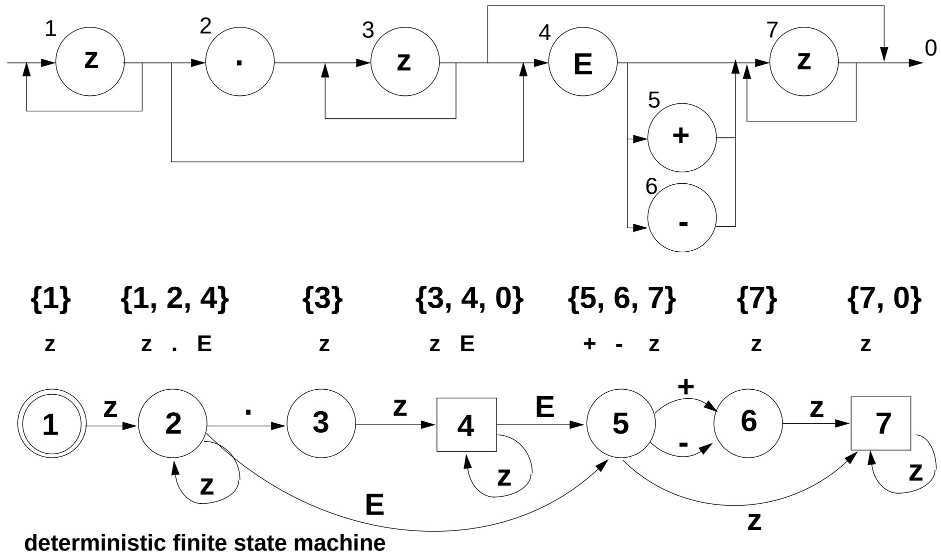


4. **repeat step 3** until no new sets of nodes can be created
5. a state q is a **final state** iff *0* is in m_q.

© 2001 bei Prof. Dr. Uwe Kastens

Example: Floating point numbers in Pascal

Syntax diagram



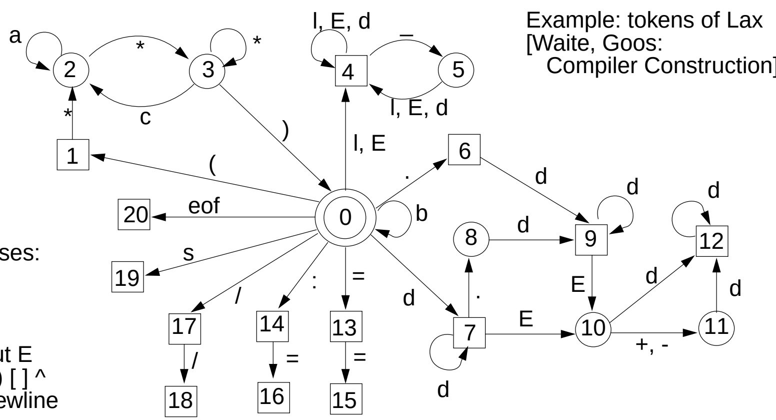
deterministic finite state machine

© 2001 bei Prof. Dr. Uwe Kastens

Composition of token automata

Construct one finite state machine for each token. Compose them forming a single one:

- **Identify the initial states of the single automata** and identical structures evolving from there (transitions with the same character and states).
- **Keep the final states of single automata distinct,** they classify the tokens.
- **Add automata for comments and irrelevant characters** (white space)



Example: tokens of Lax
[Waite, Goos:
Compiler Construction]

character classes:

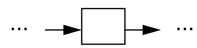
a all but *
c all but * or)
d digits
l all letters but E
s + - * < > ; ,) [ ] ^
b blank tab newline

© 2001 bei Prof. Dr. Uwe Kastens

Rule of the longest match

An automaton may contain **transitions from final states:**

When does the automaton stop?



Rule of the longest match:

- **The automaton continues as long as there is a transition with the next character.**
- **After having stopped it sets back to the most recently passed final state.**
- **If no final state has been passed an error message is issued.**

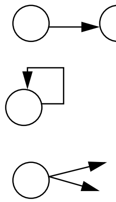
Consequence: Some kinds of tokens have to be separated explicitly.

Check the concrete grammar for tokens that may occur adjacent!

© 2001 bei Prof. Dr. Uwe Kastens

Scanner: Aspects of implementation

- **Runtime is proportional to the number of characters in the program**
- **Operations per character must be fast** - otherwise the Scanner dominates compilation time
- **Table driven** automata are too **slow:**
  Loop interprets table, 2-dimensional array access, branches
- **Directly programmed** automata is **faster;** transform **transitions into control flow:**



sequence

repeat loop

branch

- **Fast loops** for sequences of irrelevant **blanks**.
- Implementation of **character classes**:
  bit pattern or indexing - avoid slow operations with sets of characters.
- **Do not copy characters** from input buffer - maintain a pointer into the buffer, instead.

© 2006 bei Prof. Dr. Uwe Kastens

Identifier module and literal modules

- **Uniform interface for all scanner support modules:**
  Input parameters: pointer to token text and its length;
  Output parameters: syntax code, attribute

- **Identifier module encodes identifier occurrences bijective (1:1), and recognizes keywords**
  Implementation: hash vector, extensible table, collision lists

- **Literal modules for floating point numbers, integral numbers, strings**

  Variants for representation in memory:
  token text; value converted into compiler data; value converted into target data

  Caution:
  Avoid overflow on conversion!
  Cross compiler: compiler representation may differ from target representation

- **Character string memory:**
  stores strings without limits on their lengths,
  used by the identifier module and the literal modules

© 2006 bei Prof. Dr. Uwe Kastens

Scanner generators

generate the central function of lexical analysis

GLA	University of Colorado, Boulder; component of the Eli system
Lex	Unix standard tool
Flex	Successor of Lex
Rex	GMD Karlsruhe

Token specification: regular expressions

GLA	library of precoined specifications; recognizers for some tokens may be programmed
Lex, Flex, Rex	transitions may be made conditional

Interface:

GLA	as described in this chapter; cooperates with other Eli components
Lex, Flex, Rex	actions may be associated with tokens (statement sequences) interface to parser generator Yacc

Implementation:

GLA	directly programmed automaton in C
Lex, Flex, Rex	table-driven automaton in C
Rex	table-driven automaton in C or in Modula-2
Flex, Rex	faster, smaller implementations than generated by Lex

© 2001 bei Prof. Dr. Uwe Kastens